PNL-SA--20425

DE92 010829

MODELING NODE BANDWIDTH LIMITS AND THEIR EFFECT ON VECTOR COMBINING ALGORITHMS

R. J. Littlefield

Work supported by
the U.S. Department of Energy
under Contract DE-AC06-76RLO 1830

Pacific Northwest Laboratory
Richland, Washington 99352

DISCLAIMER

Modeling Node Bandwidth Limits and Their Effect on Vector Combining Algorithms

Richard J. Littlefield*
Pacific Northwest Laboratory
Richland, Washington

January 13, 1992

Abstract

Each node in a message-passing multicomputer typically has several communication links. However, the maximum aggregate communication speed of a node is often less than the sum of its individual link speeds. Such computers are called *node bandwidth limited (NBL)*. The NBL constraint is important when choosing algorithms because it can change the relative performance of different algorithms that accomplish the same task. This paper introduces a model of communication performance for NBL computers and uses the model to analyze the overall performance of three algorithms for vector combining (global sum) on the Intel Touchstone DELTA computer.[1] Each of the three algorithms is found to be at least 33% faster than the other two for some combinations of machine size and vector length. The NBL constraint is shown to significantly affect the conditions under which each algorithm is fastest.

1 Introduction

Each node in a message-passing multicomputer typically has several communication links, and algorithms are often designed to use more than one link at the same time. For example, an algorithm to shift data across a mesh may simultaneously send on one link and receive on another.

The performance of such algorithms is often estimated by assuming that the speed of each link is constant, regardless of how many links are in use. This assumption is often incorrect.

*Pacific Northwest Laboratory is operated for the U.S.Department of Energy (DOE) by Battelle Memorial Institute under contract DE-AC06-76RLO 1830. The author's address is Pacific Northwest Laboratory, P.O.Box 999, Richland, WA 99352; email rj_littlefield@pnl.gov

[1]Intel Supercomputer Systems Division, Intel Corporation, Beaverton, Oregon. The Touchstone DELTA computer is a result of specially directed efforts in support of the Concurrent Supercomputing Consortium, and is not marketed by Intel.

In actuality, conflicts for other resources can cause the effective link speed to drop when multiple links are in use. These conflicts cause the maximum aggregate communication speed of a node to be less than the sum of its individual link speeds. We say that such machines are *node bandwidth limited (NBL)* or have the *NBL constraint.*

The NBL constraint can change the relative performance of different algorithms to accomplish the same task, so the constraint should be considered when choosing algorithms. For example, given two algorithms, the one using fewer links but more communicaton steps may be faster on a machine with the NBL constraint, but slower on a machine without it.

This paper introduces a model of communication performance that reflects the NBL constraint. The model is then used to analyze the overall performance of three algorithms for vector combining (global sum) on the Intel Touchstone DELTA computer [1]. Global sum is an important operation in many applications, such as molecular dynamics using the replicated data strategy. The models and methodology presented here were developed as part of our study of alternate ways to implement protein dynamics on the DELTA.

Without the NBL constraint, one of the algorithms would hardly be worth implementing. With the NBL constraint, this algorithm becomes superior to the others over a wide range of conditions, reaching 33% faster for some combinations of machine size and vector length. It could well be a mistake to choose one of these algorithms without considering the NBL constraint.

Organization of the paper is as follows. Section 2 introduces a model of communication performance for NBL machines. Section 3 describes three algorithms for vector combining and develops models for their overall performance. Performance of the algorithms, with and without the NBL constraint, is discussed in Section 4. The summary and conclusions are found in Section 5.

2 NBL Model

The standard model for communication cost in a message-passing multicomputer is:

$$T = \alpha + \beta S$$

where T is the total time to transfer the message, α and β are constants depending on the computer hardware and operating system, and S is the message size. This model works well for most computers when each node uses only one link at a time.

When multiple messages are sent and received at nominally the same time by a single node, the standard model can break down. As a node attempts to utilize more than one communication link at the same time, conflicts can arise for other resources, such as paths to memory or processor cycles to manage the communications [2]. These conflicts cause a message to take longer than the standard model would predict.

To model this effect, we extend the cost equation to read:

$$T(L) = L\alpha + f(L)\beta S$$

where L is the number of communication links that are nominally active at the same time, and $T(L)$ is the time to send L messages of size S. That is, we assume that using multiple links does not change the overhead cost per message, and that any overlap of data transfer (if any) depends on the number of nominally active links, as indicated by $f(L)$. By definition, $f(1) = 1$.

Particularly simple forms of f are $f(L) = 1$, meaning that overlap is perfect (no bandwidth constraint), and $f(L) = L$, meaning that messages cannot overlap at all (node bandwidth equal link bandwidth). Useful, but not perfect, overlap is modeled by $1 < f(L) < L$.

3 Global Combining

3.1 Task and Approach

To illustrate dealing with the NBL constraint, we consider the task of *global vector combining*; that is, combining corresponding elements of many vectors, one per node of a distributed memory computer, and broadcasting the resulting vector to all the nodes. An example is global sum.

Global vector combining is a good task to consider because it is conceptually simple, is an important kernel operation in many distributed memory applications, and is used with widely varying data volumes and node counts. For example, in a molecular dynamics application using the replicated data strategy, updating the system state requires a global sum of vectors that may contain 10^3 to 10^5 numbers, while convergence checks in many iterative algorithms require vectors only 1 or 2 elements long. Further, the applications might run on computer systems containing from a few nodes to several hundreds or thousands of nodes.

Because of the wide variety of situations, it seems likely that using different algorithms in different situations will yield higher performance than using a single algorithm across the whole span. The fastest way to combine a single value across a thousand nodes may not be the fastest to combine a million values across a handful of nodes.

The problem is how to choose good algorithms. One approach is to implement a variety that seem plausible, and then run a wide range of experiments to choose the winners and characterize when they should be used. An alternate approach, pursued here, is to use easily constructed performance models as screening tools to decide what to implement, and then to use experimental results to refine the models so that the models can be used to choose the best algorithm for the situation at hand.

3.2 Algorithms

The three algorithms considered here are appropriate for a computer with a 2-D mesh with cut-through routing; that is, where latency is independent of distance in the absence of link conflicts. All of them work by incrementally combining data elements until a single node holds the result, from which it is broadcast. All of the algorithms are free of link conflicts;

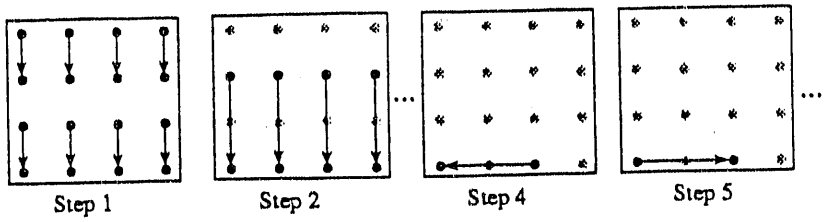

Figure 1: Communication pattern at several steps in the Tree algorithm.

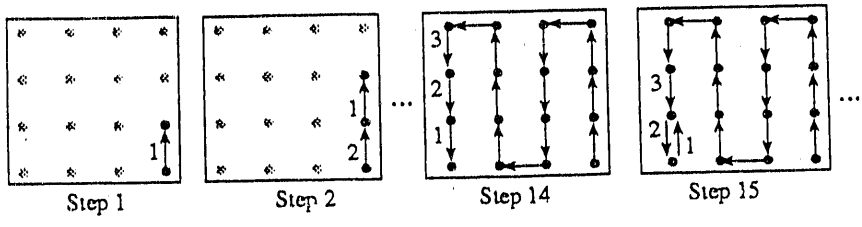

Figure 2: Communication pattern at several steps in the Snake algorithm. Numbers indicate the data block being transferred. The last diagram in this series shows block 1, now fully combined, starting to be sent back outward.

that is, no link carries more than one message in the same communication step. They differ in communication topology and whether the communications are pipelined.

Tree, illustrated in Figure 1, uses a binary tree scheme. At each step of the algorithm, half the participating nodes send all their data to the other half in a single message. After the data has arrived, the receiving nodes combine the incoming data with their own. Then half of those send all their data to the other half, and so on. When the final result is available in one node, the process is reversed to broadcast the result. Note that this algorithm does not use pipelined communications. On a hypercube, the algorithm can be mapped such that it could be pipelined without introducing link conflicts. In general this cannot be done on a mesh.

Snake, illustrated in Figure 2, uses a linear communication structure with pipelining. The data vector is processed in blocks. Let the nodes be numbered 0 to P-1 along the snake. In the first step of the algorithm, node number P-1 sends its first block of data "inward" to its neighbor, node P-2, where it is combined with that node's data. In the second step, node P-1 sends its second block of data to node P-2, while node P-2 simultaneously sends the first block of data (now partially combined) to node P-3. This process continues until node 0 receives the first block and combines it to produce a final result. In the following step, the first block of results is sent back "outward" from node 0 to 1, while subsequent blocks continue to migrate inward. Note that four links per node are active in the middle of

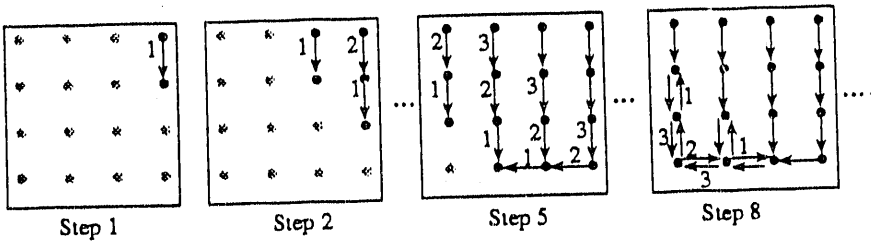

Figure 3: Communication pattern at several steps in the Fence algorithm. Numbers indicate the data block being transferred. The last diagram in this series shows blocks 1 and 2, now fully combined, being sent back outward.

pipelining.

Fence, illustrated in Figure 3, is conceptually similar to Snake, except that Fence uses a 2-D communication structure. Each block is passed down the columns and across the bottom row. Nodes in the middle of each column combine two data blocks: their own and one arriving from above. Nodes along the bottom row combine three data blocks: their own, one arriving from above, and one arriving from the right. When a block arrives at the lower left node, the final result for that block is sent back the way it came, fanning out to the right and up. Note that six links per node are active in the middle of pipelining.

3.3 Algorithm Performance Models

For simplicity, we assume that communication is not overlapped with computation. We also assume that the algorithms are loosely synchronous; that is, all of the nodes perform each step of the algorithm at the same time. (This assumption is not strictly justified since there is no global synchronization. In fact, nodes can get substantially out of sync with our pictures, a point we will touch on in a later section.) With these assumptions, the total execution time is the sum of the communication and computation times for the busiest node in each step. Since these algorithms pass the same amount of data on each link (if any is passed at all), we will also assume that the busiest node is the one with the most active links.

With these assumptions, writing the models is a straightforward if tedious task of enumerating what communication and computation is done at each step in the algorithm. The derivation and resulting models are shown in the Appendix.

4 Performance Impact of the NBL Constraint

4.1 Theory

Our model of the NBL constraint is that it effectively reduces the speed of each link when more than one link is used at the same time. This constraint has three main effects.

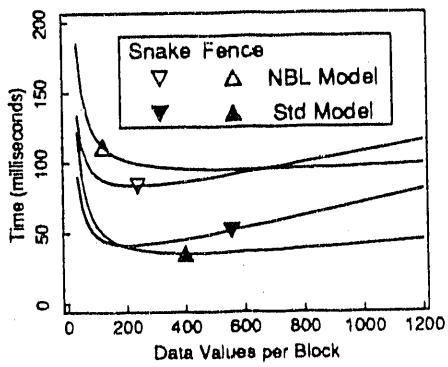

Figure 4: Effect of the NBL constraint on the relative performance and optimum block sizes for the Snake and Fence algorithms.

- Algorithms using more than one link run slower than if the effective link speed were constant.
- Algorithms that use more links at once are affected worse than algorithms that use fewer links.
- The ratio of startup cost to transfer time $(\alpha/(f(L)\beta)$ is changed, which can alter parameters like the optimum block size for pipelined communications.

These effects are illustrated in Figures 4 and 5, which compare the following two sets of model assumptions:

Standard Model	$f(L) = 1$
Nominal NBL Model	$f(L) = L$

Figure 4 shows execution time of the Snake and Fence algorithms as a function of block size for a typical case. Note that the NBL model predicts significantly slower execution, a shift in the optimum block size, and a reversal of which algorithm is faster.

Figure 5 shows the fastest algorithm (Tree, Fence, or Snake) as a function of vector length and mesh size. Under both models, the Tree algorithm is fastest for short vectors. Without the NBL constraint, the Fence algorithm is fastest almost everywhere else, while Snake wins only in a small regime with very long vectors on just a few nodes. In contrast, with the NBL constraint, Snake is fastest over a large regime including both moderately long vectors and many nodes.

These results can be understood in terms of the number of steps and asymptotic (long vector) throughput of each algorithm. Tree has the fewest steps, but also has the lowest

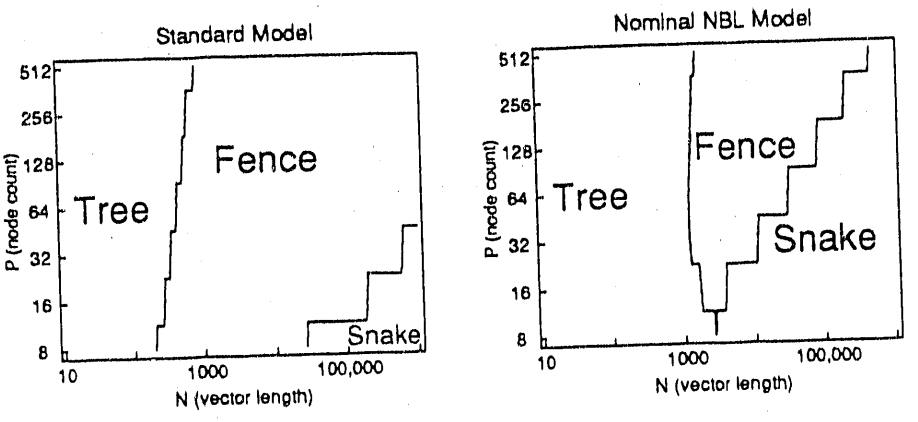

Figure 5: Effect of the NBL constraint on the regimes where each of the three algorithms is fastest.

throughput because of the lack of pipelining. Fence is intermediate in both number of steps and throughput, and Snake has the most steps but the highest throughput. The NBL constraint penalizes both Fence and Snake, but affects Fence worse because it uses more links at once (six for Fence versus four for Snake). The effect is to enlarge the regimes in which Tree and Snake are fastest.

4.2 Empirical Results

This section describes a series of experiments performed on the Intel Touchstone DELTA computer [1]. As shown in Figure 6, the DELTA has a 2-D mesh topology. Each node of the mesh consists of a router module that has independent links in both directions to each of its 4 nearest neighbors, plus another pair of links to the node processor. All data coming into and out of the node processor share that pair of links and their associated queueing hardware. In addition, data movement to and from the queues is done by the node processor, leading to the possibility of processor saturation. This description suggests that the DELTA probably has some sort of NBL constraint. However, it is not easy to identify which of the factors would be practical limits, and thus it was not clear *a priori* how stringent the constraint would be.

Table 1 shows the results of a testjig program designed to investigate the NBL constraint in a simple setting. The program runs on a 3×3 mesh, with the central node communicating with 1, 2, or 3 of its neighbors. Four cases were measured: central node just receiving from one neighbor (L=1) and central node exchanging (both sending and receiving) with one, two, and three neighbors (L=2, L=4, L=6). The results indicate that for long messages the machine is almost perfectly NBL constrained ($f(L) = L$), but that shorter messages appear to have some overlap ($f(L) < L$).

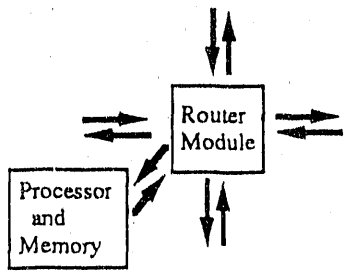

Figure 6: Topology of the Intel Touchstone DELTA computer. Each router module has independent links in both directions to each of its four nearest neighbors, plus another pair of links to the node processor.

	L = 1		L = 2			L = 4			L = 6		
S	T	dT	T	dT	$f(2)$	T	dT	$f(4)$	T	dT	$f(6)$
60	74		129			259			345		
		65		106	1.6		139	2.1		230	3.8
120	139		235			398			575		
		154		267	1.7		435	2.8		647	4.2
300	293		502			833			1222		
		232		440	1.9		900	3.9		1356	.8
600	525		942			1733			2578		

Table 1: Values of $f(\mathrm{L})$ as determined by a testjig program. S is the number of double precision values per block; T is total time in microseconds, dT is the difference of adjacent T's; $f(\mathrm{L}) = \mathrm{dT(L)} \;/\; \mathrm{dT(1)}$.

All three global combining algorithms were also implemented and measured. Test cases included all combinations of three mesh sizes (4×4, 16×16, and 16×32), 5 vector lengths (L=10^2, 10^3, 10^4, 10^5, and 5×10^5), and a range of block sizes (S=20 to 600).

In practice, the Fence and Snake algorithms consistently perform a little better than predicted by the nominal $f(L) = L$. Qualititatively, an effect like this would be expected, since the measured $f(L)$ shown in Table 1 are smaller than nominal values. However, it is conceivable that the effect could be due to other causes. One way to check would be to directly incorporate the measured $f(L)$ into the model. However, since $f(L)$ apparently varies depending on block size, this would require more empirical work to fully characterize the variation of $f(L)$.

A simpler indication can be obtained by fitting the general NBL model to the empirical data by treating the $f(L)$ as parameters. This produces the following result:

Fitted NBL Model: $f(2) = 1.1$
$f(3) = 1.3$
$f(4) = 3.9$
$f(6) = 5.1$

Figure 7 shows a sample of experimental data illustrating the accuracy of the fitted NBL model and relative performance of the three algorithms. It is apparent that the fitted NBL model is accurate over a wide range of N and P. This figure also helps in understanding the transition regime, where the fastest algorithm is first Tree, then Fence, then Snake, as the vector length is increased. The final two cases, $P \doteq 16$, N=10^5 and N=5×10^5, illustrate the regime with very long vectors, where Snake is asymptotically up to 33% faster than Fence.

Although the fitted NBL model is accurate, we must be cautious in assigning meaning to the $f(L)$ obtained in this way. On the surface, the numbers seem to imply that two or three links can be used with good overlap ($f(L) \approx 1$), while with four or more links there is little if any overlap ($f(L) \approx L$). This seems unlikely and is inconsistent with the testjig measurements shown in Table 1. We suspect that the real explanation lies in the dynamics of the Fence and Snake algorithms. The fitted $f(4)$ and $f(6)$ values describe the period during which the pipelines are full, constraining the system to work synchronously. However, the fitted $f(2)$ and $f(3)$ values describe the early and late stages of each algorithm, while the pipelines are filling and draining. We speculate that these stages violate the assumption of synchronization, and that data blocks become separated in the mesh in such a way that only a single communication link is actually in use when the model says that two or three should be. We plan to study this phenomenon more closely in future work. Until the issue is resolved, it seems prudent to use the nominal NBL model to evaluate unimplemented algorithms.

The effectiveness of using the nominal NBL model can be seen by comparing Figures 8 and 5. Figure 8 shows the regimes where each algorithm is observed to be best, while Figure 5 shows the predictions of both nominal NBL and the standard model. It is apparent that the nominal NBL model provided a more accurate evaluation.

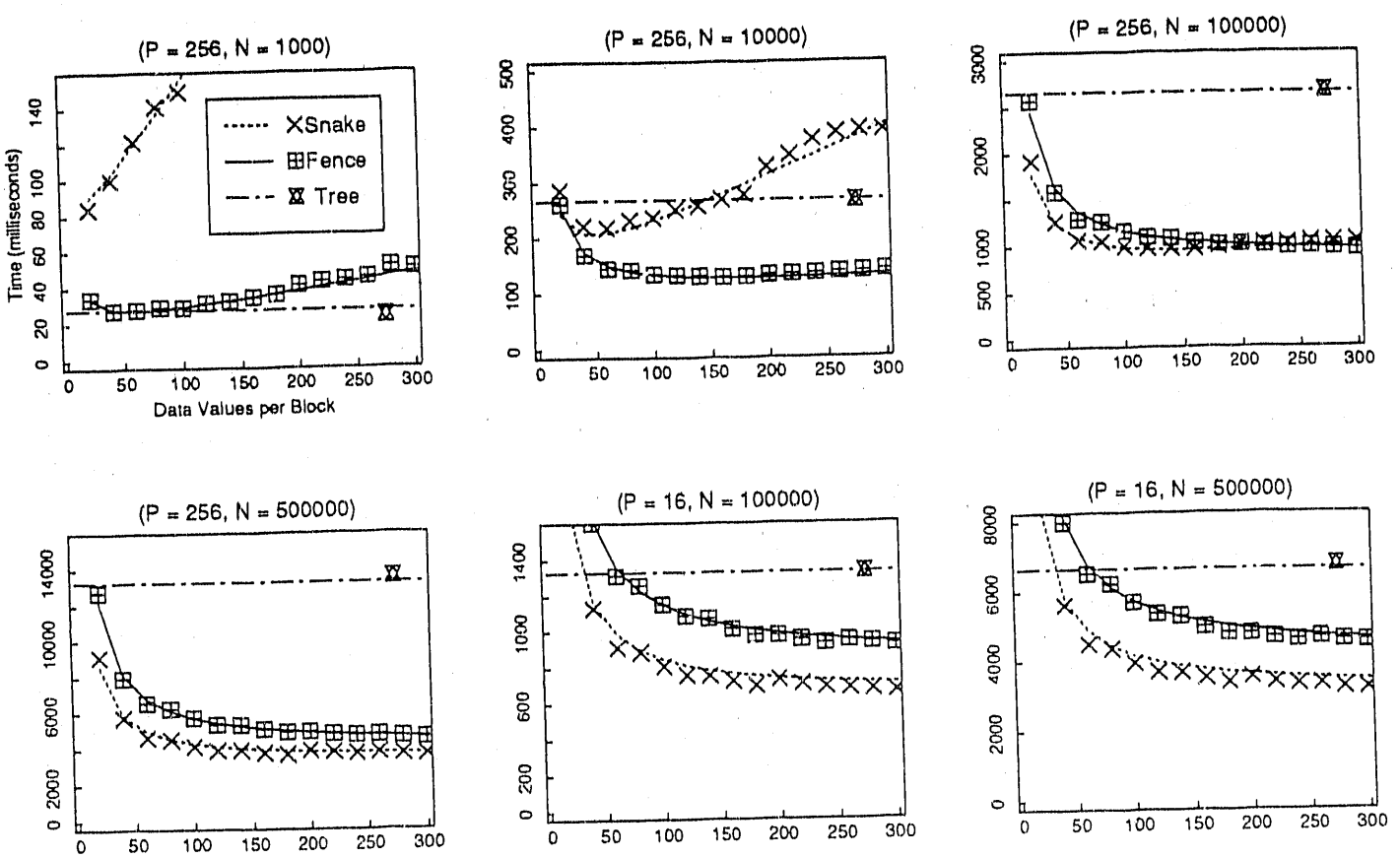

Figure 7: Experimental data compared to the fitted NBL model, over the transition where each algorithm is faster for certain vector lengths.

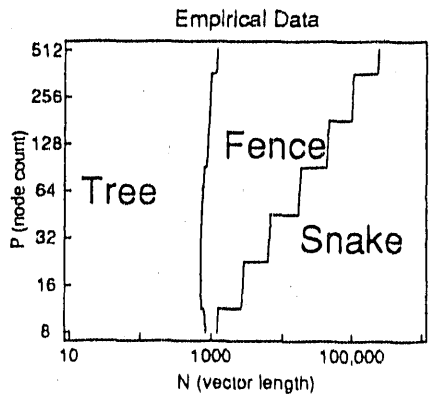

Figure 8: Regimes where each of the three algorithms is fastest, based on empirical data.

5 Summary and Conclusions

We have introduced a model of communication performance for *node bandwidth limited (NBL)* computers, and have used that model to analyze the overall performance of three algorithms for global vector combining on a 2-D mesh computer. The NBL constraint has been shown to be important in determining the regimes in which each of the algorithms is fastest. With the NBL constraint, each of the three algorithms is the best by a significant margin in useful regimes. Empirical results from the Intel Touchstone DELTA computer have confirmed the theoretical models and indicate that the DELTA behaves as an NBL machine. These results will be important in choosing and designing further algorithms for the DELTA and machines with similar characteristics.

6 Acknowledgements

This research was supported by the U.S. Department of Energy (DOE). Access to the Intel Touchstone DELTA System of the Concurrent Supercomputing Consortium was provided by the DOE. DE-AC06-76RLO 1830

7 Appendix

7.1 Derivation of Performance Models

The following symbols are defined:

W width of the mesh (number of nodes per row).

H height of the mesh (number of nodes per column).

P total number of nodes ("processors"), P = HW.

α, β, f, L as per the NBL model.

c_2 computational cost, per element, of combining two vectors (e.g., $v1 = v1 + v2$).

c_3 computational cost, per element, of combining three vectors at the same time (e.g., $v1 = v1 + v2 + v3$).

N number of elements per vector.

S size of each block of data (for pipelined communication schemes).

B number of blocks of data ($B = \lceil N/S \rceil$).

Tree: This algorithm does no pipelining and each node uses only a single communication link per step.

$$T = (\log_2(W) + \log_2(H)) * (2(\alpha + f(1)\beta N) + c_2 N)$$

Snake: This algorithm uses different numbers of links in various steps. Assuming that $B \geq 3$, stepping through the algorithm shows the following sequence of activity:

Number of steps	Number of active links (L)	Comment
1	1	Node P-1 sends, P-2 receives
P-2	2	Some node sends and receives partially combined data (PCD)
1	3	Node 1 also receives final result
B-3	4	Node 1 sends and receives data and final results
1	3	Node 1 sends PCD, sends and receives final results
P-2	2	Some node sends and receives final results
1	1	Node P-2 sends P-1 final results

The total execution time is:

$$\begin{aligned} T = \quad & 1 \quad * (1 * (\alpha + f(1)\beta S) + c_2 S) + \\ & \text{(P-2)} * (2 * (\alpha + f(2)\beta S) + c_2 S) + \\ & 1 \quad * (3 * (\alpha + f(3)\beta S) + c_2 S) + \\ & \text{(B-3)} * (4 * (\alpha + f(4)\beta S) + c_2 S) + \\ & 1 \quad * (3 * (\alpha + f(3)\beta S) + c_2 S) + \\ & \text{(P-2)} * (2 * (\alpha + f(2)\beta S)) + \\ & 1 \quad * (1 * (\alpha + f(1)\beta S)) \end{aligned}$$

The optimum block size S under this model can be determined by solving $dT/dS = 0$. (Symbolic math packages are good for this.)

$$S_{opt} = (N\alpha/((P-2)(c_2 + 2\beta f(2)) + 2\beta f(3) - 3\beta f(4)))^{1/2}$$

Fence: Following the same procedure as with Snake, the timeline is:

Number of steps	Number of active links (I)	Comment
1	1	First block starts down from top
H-1	2	First block continues down, turns corner
W-2	3	3-way combining along bottom row
1	4	Result turns around at node 0,0
B-3	6	3-way combining and 2-way fanout along bottom row
1	4	Final block goes to node 0,0
W-2	3	Results return along bottom row.
H-1	2	Last result block turns corner and goes up
1	1	Last block enters upper right node

The total execution time is:

$$\begin{aligned} T = \quad & 1 \quad * (1 * (\alpha + f(1)\beta S) + c_2 S) + \\ & \text{(H-1)} * (2 * (\alpha + f(2)\beta S) + c_2 S) + \\ & \text{(W-2)} * (3 * (\alpha + f(3)\beta S) + c_3 S) + \\ & 1 \quad * (4 * (\alpha + f(4)\beta S) + c_3 S) + \\ & \text{(B-3)} * (6 * (\alpha + f(6)\beta S) + c_3 S) + \\ & 1 \quad * (4 * (\alpha + f(4)\beta S) + c_3 S) + \end{aligned}$$

$$
\begin{aligned}
&(\text{W-2}) * (3 * (\alpha + f(3)\beta S)) + \\
&(\text{H-1}) * (2 * (\alpha + f(2)\beta S)) + \\
&\quad 1 \quad * (1 * (\alpha + f(1)\beta S))
\end{aligned}
$$

and the optimum block size is:

$$
\begin{aligned}
S_{opt} \;=\; & (6N\alpha/[H(c_2 + 2\beta f(2)) + W(c_3 + 2\beta f(3)) \\
& - 3c_3 + \beta(2f(1) - 2f(2) - 4f(3) + 2f(4) - 3f(6))])^{1/2}
\end{aligned}
\tag{1}
$$

7.2 DELTA System Parameters

The results reported here used DELTA system parameters determined by a variety of testjig programs run at software revision X020 using the Intel communication routines csend and crecv. The link speed (β) was deliberately reduced by a factor of 2X during this test period to avoid intermittent hardware problems. The value reported here should not be taken to represent system performance in production mode.

α = 54 μsec
β = 1.54 μsec/double (5.2 Mbytes/sec)
c_2 = .25 μsec (4.0 MFLOPS)
c_3 = .37 μsec (5.3 MFLOPS)

References

[1] Intel Corp. *A Touchstone DELTA System Description.* Intel Supercomputer Systems Division, Beaverton, Oregon., 1991.

[2] T. H. Dunigan. Performance of the INTEL iPSC/860 and NCUBE 6400 hypercubes. Technical Report ORNL/TM-11790, Oak Ridge Nat. Lab. April 1991.

DATE
FILMED
5/01/92